CS 245: Database System Principles

Notes 03: Disk Organization

Hector Garcia-Molina

Topics for today

- How to lay out data on disk
- How to move it to memory

What are the data items we want to store?

- a salary
- a name
- a date
- a picture

⇨ What we have available: Bytes

To represent:

- Integer (short): 2 bytes
  e.g., 35 is
  00000000 00100011
- Real, floating point
  n bits for mantissa, *m* for exponent….

To represent:

- Characters
  → various coding schemes suggested, most popular is ascii

  Example:
  A: 1000001
  a: 1100001
  5: 0110101
  LF: 0001010

To represent:

- Boolean
  e.g., TRUE 1111 1111
  FALSE 0000 0000
- Application specific
  e.g., RED → 1 GREEN → 3
  BLUE → 2 YELLOW → 4 …

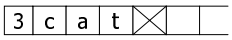
⇨ Can we use less than 1 byte/code?
Yes, but only if desperate...

To represent:

- Dates
  e.g.: - Integer, # days since Jan 1, 1900
  - 8 characters, YYYYMMDD
  - 7 characters, YYYYDDD
  (not YYMMDD! Why?)
- Time
  e.g. - Integer, seconds since midnight
  - characters, HHMMSSFF

To represent:

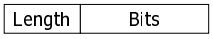
- String of characters
  - Null terminated
    e.g., 

  - Length given
    e.g., 

  - Fixed length

To represent:

- Bag of bits



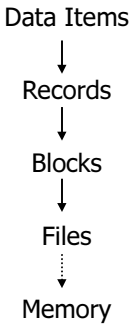
Key Point

- Fixed length items
- Variable length items
  - usually length given at beginning

Also

- Type of an item: Tells us how to interpret (plus size if fixed)

Overview



Record - Collection of related data items (called FIELDS)

E.g.: Employee record:
name field,
salary field,
date-of-hire field, ...

Types of records:

- Main choices:
  - FIXED vs VARIABLE FORMAT
  - FIXED vs VARIABLE LENGTH

Fixed format

A SCHEMA (not record) contains following information
- # fields
- type of each field
- order in record
- meaning of each field

Example: fixed format and length

Employee record
(1) E#, 2 byte integer
(2) E.name, 10 char.
(3) Dept, 2 byte code

Schema

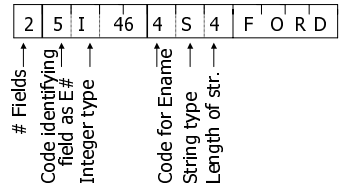
55	smith	02
83	jones	01

Records

Variable format

- Record itself contains format "Self Describing"

Example: variable format and length



Field name codes could also be strings, i.e. TAGS

Variable format useful for:

- "sparse" records
- repeating fields
- evolving formats

⋯▸ But may waste space...

- EXAMPLE: var format record with repeating fields
  Employee → one or more → children

3	E_name: Fred	Child: Sally	Child: Tom

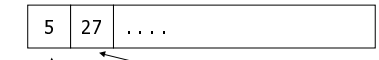
Note: Repeating fields does not imply
- variable format, nor
- variable size

John	Sailing	Chess	--

- Key is to allocate maximum number of repeating fields (if not used → null)

☆ Many variants between fixed - variable format:

Ex. #1: Include record type in record



record type tells me what to expect (i.e. points to schema)

record length

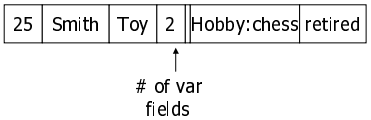
Record header - data at beginning that describes record

May contain:
- record type
- record length
- time stamp
- other stuff ...

Ex #2 of variant between FIXED/VAR format

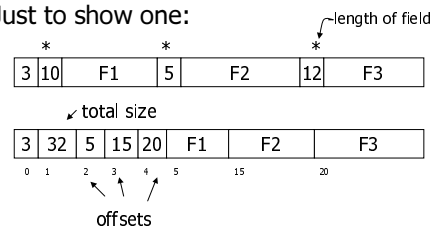
- Hybrid format
  – one part is fixed, other variable

E.g.: All employees have E#, name, dept other fields vary.



☆ Also, many variations in internal organization of record

Just to show one:



Question:

We have seen examples for

* Fixed format and length records
* Variable format and length records

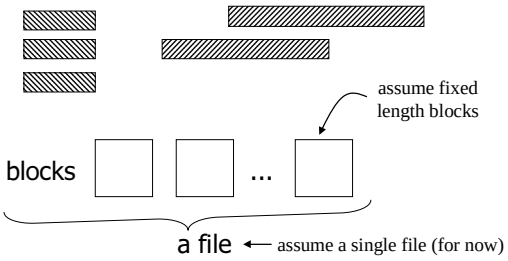
(a) Does fixed format and variable length make sense?

(b) Does variable format and fixed length make sense?

Other interesting issues:

- Compression
  - within record - e.g. code selection
  - collection of records - e.g. find common patterns
- Encryption

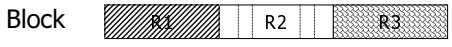
Next: placing records into blocks



Options for storing records in blocks:

(1) separating records
(2) spanned vs. unspanned
(3) mixed record types – clustering
(4) split records
(5) sequencing
(6) indirection

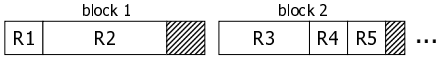
(1) Separating records



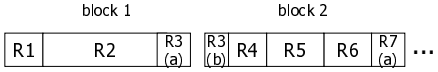
(a) no need to separate - fixed size recs.
(b) special marker
(c) give record lengths (or offsets)
- within each record
- in block header

(2) Spanned vs. Unspanned

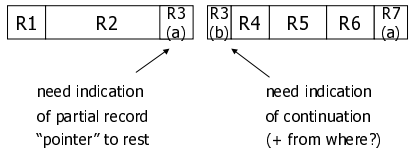
- Unspanned: records must be within one block



- Spanned



With spanned records:

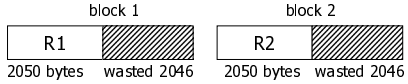


Spanned vs. unspanned:

- Unspanned is much simpler, but may waste space…
- Spanned essential if
  record size > block size

Example

10^6 records
each of size 2,050 bytes (fixed)
block size = 4096 bytes

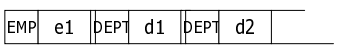


- Total wasted = 2×10^9 Utiliz = 50%
- Total space = 4×10^9

(3) Mixed record types

- Mixed - records of different types
  (e.g. EMPLOYEE, DEPT)
  allowed in same block

e.g., a block



Why do we want to mix?
Answer: CLUSTERING

Records that are frequently accessed together should be in the same block

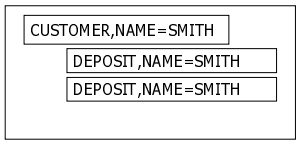
Compromise:

No mixing, but keep related records in same cylinder ...

Example

Q1: select A#, C_NAME, C_CITY, …
from DEPOSIT, CUSTOMER
where DEPOSIT.C_NAME =
CUSTOMER.C.NAME

a block



- If Q1 frequent, clustering good
- But if Q2 frequent

Q2: SELECT *
FROM CUSTOMER

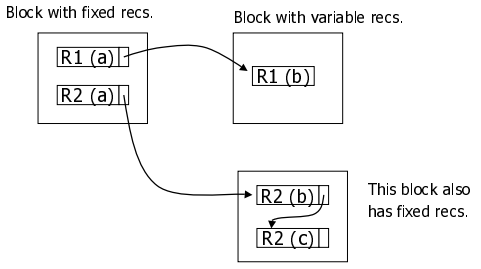
CLUSTERING IS COUNTER PRODUCTIVE

(4) Split records

Typically for hybrid format

Fixed part in one block

Variable part in another block



Question

What is difference between
- Split records
- Simply using two different record types?

(5) Sequencing

- Ordering records in file (and block) by some key value

  Sequential file ($\Rightarrow$ sequenced)

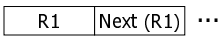
Why sequencing?

Typically to make it possible to efficiently read records in order

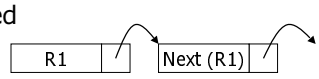
(e.g., to do a merge-join — discussed later)

Sequencing Options

(a) Next record physically contiguous



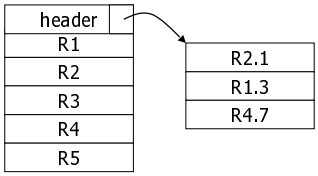
(b) Linked



Sequencing Options

(c) Overflow area

Records in sequence



(6) Indirection

- How does one refer to records?

Many options:

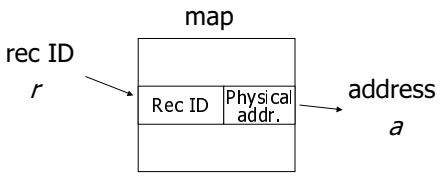
Physical $\longleftrightarrow$ Indirect

☆ Purely Physical

E.g., Record Address or ID = Device ID, Cylinder #, Track #, Block # (Block ID), Offset in block

☆ Fully Indirect

E.g., Record ID is arbitrary bit string



Tradeoff

Flexibility ⟷ Cost

to move records of indirection

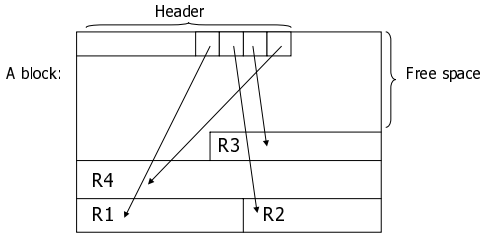
(for deletions, insertions)

Physical ⟷ Indirect

↑

Many options in between …

Ex #1 Indirection in block



Block header - data at beginning that describes block

May contain:

- File ID (or RELATION or DB ID)
- This block ID
- Record directory
- Pointer to free space
- Type of block (e.g. contains recs type 4; is overflow, …)
- Pointer to other blocks "like it"
- Timestamp …

Ex. #2 Use logical block #'s understood by file system

REC ID → File ID
Block #
Record # or Offset

File system map may be "Semi-physical"...

File F1: physical address of block 1
table of bad blocks:
B57 → XXX
B107 → YYY

Rest can be computed via formula...

Num. Blocks: 20
Start Block: 1000
Block Size: 100
Bad Blocks:
3 → 20,000
7 → 15,000

File DEFINITION

Where is Block # 2?

Where is Block # 3?

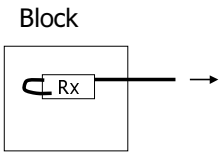
Options for storing records in blocks

(1) Separating records
(2) Spanned vs. Unspanned
(3) Mixed record types - Clustering
(4) Split records
(5) Sequencing
(6) Indirection

Other Topics

(1) Insertion/Deletion
(2) Buffer Management
(3) Comparison of Schemes

Deletion



Options:

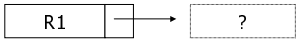
(a) Immediately reclaim space
(b) Mark deleted
- May need chain of deleted records (for re-use)
- Need a way to mark:
  - special characters
  - delete field
  - in map

☆ As usual, many tradeoffs...

- How expensive is to move valid record to free space for immediate reclaim?
- How much space is wasted?
  - e.g., deleted records, delete fields, free space chains,...

Concern with deletions

Dangling pointers

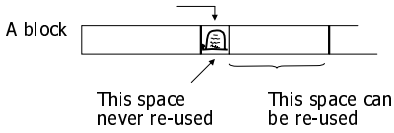


Solution #1: Do not worry

Solution #2: Tombstones

E.g., Leave "MARK" in map or old location

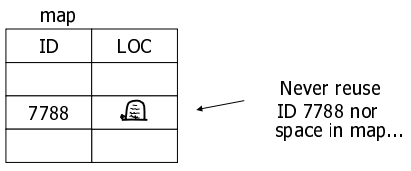
- Physical IDs



Solution #2: Tombstones

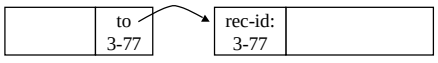
E.g., Leave "MARK" in map or old location

- Logical IDs



Solution #3 (?):

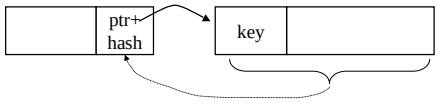
- Place record ID within every record
- When you follow a pointer, check if it leads to correct record



Does this work???
If space reused, won't new record have same ID?

Solution #4 (?):

- To point, use (pointer + hash) or (pointer + key)?



- What if record modified???

Insert

Easy case: records not in sequence

→ Insert new record at end of file or in deleted slot

→ If records are variable size, not as easy...

Insert

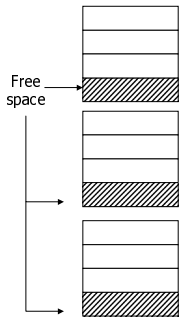
Hard case: records in sequence

→ If free space "close by", not too bad...

→ Or use overflow idea...

Interesting problems:

- How much free space to leave in each block, track, cylinder?
- How often do I reorganize file + overflow?



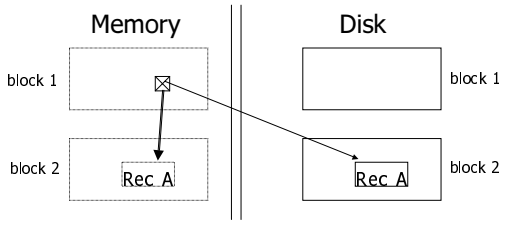
Buffer Management

- DB features needed
- Pinned blocks
- Forced output (flush)
- Why LRU may be bad
- Double buffering
- Swizzling

Read Textbook!

in Notes02

Swizzling



One Option:

Translation Table

DB Addr	Mem Addr
Rec-A	Rec-A-inMem

Another Option:

In memory pointers - need "type" bit

- ⊠ to disk
- M to memory

Swizzling

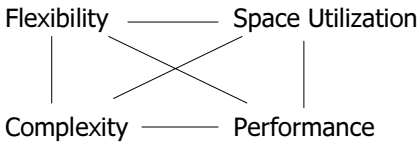
- Automatic
- On-demand
- No swizzling / program control

Comparison

- There are 10,000,000 ways to organize my data on disk…

  Which is right for me?

Issues:



☆ To evaluate a given strategy, compute following parameters:

-> space used for expected data

-> expected time to

- fetch record given key
- fetch record with next key
- insert record
- append record
- delete record
- update record
- read all file
- reorganize file

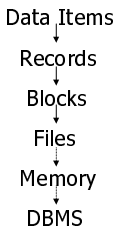
Example

How would you design Megatron 3000 storage system? (for a relational DB, low end)

- Variable length records?
- Spanned?
- What data types?
- Fixed format?
- Record IDs ?
- Sequencing?
- How to handle deletions?

Summary

- How to lay out data on disk



Next

How to find a record quickly, given a key